EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Optimizing quasi-cyclic spatially coupled LDPC codes by eliminating harmful objects

Massimo Battaglioni[1*†], Franco Chiaraluce[1†], Marco Baldi[1†], Michele Pacenti[2†] and David G. M. Mitchell[3†]

†Massimo Battaglioni, Franco Chiaraluce, Marco Baldi, Michele Pacenti and David G. M. Mitchell have contributed equally to this work.

*Correspondence: m.battaglioni@univpm.it

[1] Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche and CNIT, Via Brecce Bianche 12, 60131 Ancona, Italy
[2] Department of Electrical and Computer Engineering, University of Arizona, 1230 E Speedway Blvd, Tucson, AZ 85719, USA
[3] Klipsch School of Electrical and Computer Engineering, New Mexico State University, 1125 Frenger St, Las Cruces, NM 88011, USA

Abstract

It is well known that some harmful objects in the Tanner graph of low-density parity-check (LDPC) codes have a negative impact on their error correction performance under iterative message-passing decoding. Depending on the channel and the decoding algorithm, these harmful objects are different in nature and can be stopping sets, trapping sets, absorbing sets, or pseudocodewords. Differently from LDPC block codes, the design of spatially coupled LDPC codes must take into account the semi-infinite nature of the code, while still reducing the number of harmful objects as much as possible. We propose a general procedure, based on *edge spreading*, enabling the design of good quasi-cyclic spatially coupled LDPC (QC-SC-LDPC) codes. These codes are derived from quasi-cyclic LDPC (QC-LDPC) block codes and contain a considerably reduced number of harmful objects with respect to the original QC-LDPC block codes. We use an efficient way of enumerating harmful objects in QC-SC-LDPCCs to obtain a fast algorithm that spans the search space of potential candidates to select those minimizing the multiplicity of the target harmful objects. We validate the effectiveness of our method via numerical simulations, showing that the newly designed codes achieve better error rate performance than codes presented in previous literature.

Keywords: Convolutional codes, Cycles, Harmful objects, Iterative decoding, LDPC codes, Spatially coupled codes, Trapping sets

1 Introduction

Low-density parity-check (LDPC) convolutional codes or (SC-LDPCCs), which have been introduced in [1] as the convolutional counterpart of LDPC block codes [2], have attracted much interest from the scientific community in recent years. Efforts have been expended in order to make them suitable for wireless communications [3, 4]. This is due to the fact that while LDPC block codes are known to approach the channel capacity [5], terminated SC-LDPCCs have been shown able to achieve the channel capacity under belief propagation-based iterative decoding, for a wide range of channels, owing to the convolutional structure of their parity-check matrix, which triggers the well-known threshold saturation phenomenon [6–8].

The performance of such iterative decoders, however, is significantly affected by the code representation, in addition to its structural properties. In fact, iterative decoding is performed on a bipartite graph known as Tanner graph [9], which has a small

number of edges corresponding to the small number of ones in the sparse parity-check matrix of an LDPC code. It is well known that some properties of the Tanner graph, like short cycles, negatively affect the performance of iterative decoders. Other well-known harmful objects in the Tanner graph are called *stopping sets* [10], which determine the failure of iterative decoding algorithms over the binary erasure channel. Different families of harmful objects are of interest to characterize the performance of these decoders over different channels, such as *trapping sets* for the additive white Gaussian noise (AWGN) channel [11]. A special subclass of trapping sets, known as *absorbing sets*, was shown to strongly affect the performance of bit-flipping iterative decoders [12]. In many cases, there is a strong connection between these harmful objects and cycles in the Tanner graph of a code [13–15]. In particular, in [13, 14] it has been shown that starting from a cycle, or a cluster of cycles in the Tanner graph of a regular or irregular LDPC block code, any trapping set can be obtained by means of some graph expansion technique.

The presence of the aforementioned harmful objects may yield an important degradation of the error correction performance of an iteratively decoded code, especially in the so-called *error-floor* region, i.e., the high signal-to-noise ratio region. Although the effect of harmful objects on the performance of iterative decoders depends on both the specific decoding algorithm and the channel, reducing the number of harmful objects generally is an important target to optimize the performance of LDPC codes. Two main approaches can be adopted to this end: i) avoiding or reducing harmful objects during code design [15–17] and ii) modifying the decoding algorithm in such a way as to be less affected by harmful objects [18, 19]. Solutions combining both approaches have also been proposed [20]; however, the former approach has generally received more attention than the latter, because working at design stage provides more degrees of freedom than acting on the decoding algorithm and thus opens more avenues for performance optimization. For this reason, we focus on the former approach to optimize the performance of SC-LDPCCs.

One common method for the design of SC-LDPCCs starts from LDPC block codes and exploits an *edge-spreading* procedure to achieve a convolutional version of the block code [21]. This is a generalization of the unwrapping technique introduced in [1, 22]. When SC-LDPCCs are designed following this approach, the harmful objects of the SC-LDPCC derive from the corresponding objects of the underlying LDPC block code, and their multiplicity depends on the adopted edge-spreading technique. Thus, for a given LDPC block code, the problem of designing good SC-LDPCCs from the harmful objects reduction standpoint translates into the problem of finding good spreading matrices. For the sake of brevity, in the rest of the paper we denote quasi-cyclic SC-LDPCCs (QC-SC-LDPCCs) obtained from quasi-cyclic low-density parity-check (QC-LDPC) block codes by edge spreading as QC-SC codes. The matrix according to which spreading is performed is called *spreading matrix* and denoted as $\mathbf{B}$. The edge-spreading procedure can also be displayed graphically by means of *protographs*, which are Tanner graphs with a relatively small number of variable and check nodes. We remark that the matrix and the graphical interpretations are completely equivalent. Protograph-based codes with excellent asymptotic and finite-length performance have been designed; see, for example, [23–25].

1.1 Previous work

A straightforward approach to the aforementioned optimization problem is that of exhaustively exploring the space of spreading matrices, which obviously poses complexity issues. Some previous works have addressed the problem of reducing the search space of candidate spreading matrices. A basic approach was proposed in [26], where the authors minimize the number of (3, 3) absorbing sets (ASs) in (m, n)-regular (AB) QC-SC codes with $m = 3$, obtained through cutting vectors, which are a subclass of spreading matrices (see [27] for further details). Such an approach is very efficient, since it relies on an integer optimization procedure, but the spanned search space is very small. In fact, the cutting vectors, as defined in [26, 27], only support QC-SC codes with memory $m_s = 1$ and they only cover $\binom{n}{3}$ spreading matrices, instead of the total 8^n. This yields a non-negligible probability that some optimal matrices are left out of the search.

In [28], a guided random search is used to find optimal spreading matrices for (m, n)-regular AB QC-SC codes with $m = 3$, where a small subset of all the possible columns is considered in such a way that the spreading matrix is "balanced." Although this approach can result in a reasonably fast search, especially if the subset contains a small number of elements, it is expected to be suboptimal, since it spans a search space which is considerably smaller than the whole space, without considering any optimization criterion. In particular, when $m_s = 1$ ($m_s = 2$, respectively), given that $m = 3$, the guided random search in [28] includes 5^n out of the total 8^n (27^n, respectively) possible spreading matrices.

The method proposed in [29] is similar to that proposed in [28]. In fact, only a subset of all the possible spreading matrices is considered, such that each possible entry of $\mathbf{B}$, which is in $\{0, 1, \ldots, m_s\}$, appears $\frac{nm}{m_s+1}$ times. This also results in a type of balanced spreading matrix. Nevertheless, in this case, the search may again be non-optimal, since a large number of spreading matrices are excluded a priori (more details on the computation of the exact number of candidates are provided in [29] and are omitted here for saving space).

The approach in [30] relies on a searching algorithm which is not described in the original paper. For this reason, we are not able to estimate the number of candidates it considers. Nevertheless, the authors of [30] mention that the search is limited, implying that it suffers from similar limitations as the methods proposed in [28, 29]. Finally, in [31, 32], a two-stage procedure is proposed: First, an optimal partitioning is applied; then, the shifts of the circulant matrices are adapted in order to minimize the number of harmful objects. However, the adaption of the shifts alters the initial code. Our focus is on the first stage of the approach proposed in [31], since our method assumes that the underlying block code is fixed. Under such an assumption, our method outperforms the approach proposed in [31], as we will show in Sect. 4. Combining our method with a preliminary optimization stage of the starting block code is an avenue for further optimization, but is beyond the scope of this paper.

As summarized above, most of the previous approaches consider only certain code structures and certain harmful objects to make their search feasible. Namely, for (m, n)-regular codes, only $(3, 3(m-2))$ trapping sets (i.e., cycles of length 6 for the

considered codes), whose minimization is shown to be beneficial also to the minimization of (4, 2), (4, 4) and (5, 3) trapping sets (among others), are the target of the minimization algorithms in [26–31]. In [32], $(4, 4(m-2))$ trapping sets are also targeted. Furthermore, many possible solutions are excluded a priori. In [28, 29, 31, 33], for example, it is shown that the multiplicity of harmful objects can be significantly reduced by increasing the memory of SC-LDPCCs. However, the computational complexity of the aforementioned approaches makes them unsuitable for the design of SC-LDPCCs with moderate and large memories.

A quite different approach has been proposed in [34]. In that work, the authors deal with the construction and performance of SC-LDPCCs with finite memory (expressed as smoothing parameter w, such that $w = m_s - 1$), showing through asymptotic analysis that non-uniform coupling yields to faster decoding convergence with respect to uniform coupling. Such a method differs from ours in that it is tailored to the binary erasure channel (BEC), relies on asymptotic analysis (and therefore does not take into account the harmful objects influencing finite-length performance), and is restricted to small values of the memory.

To the best of the authors' knowledge, except for the method concurrently proposed in [35], a general scheme enabling the construction of optimized QC-SC codes (in terms of minimization of harmful objects) with moderate memories is still missing in the literature.

1.2 Our contribution

We propose an algorithm that, given any QC-LDPC block code, exploits a smart strategy to construct a QC-SC code containing the smallest possible number of target objects (or combinations of objects), which are the most harmful ones for the given channel and decoding algorithm. We generalize the algorithm proposed in [36], which takes the target length of cycles as input without distinguishing those that are actually harmful from those that are not. The approach in [36] is efficient when the most harmful objects are cycles, or combination of a few short cycles, and if their multiplicity is relatively small. However, when the most harmful objects have a more complicated structure, targeting all the cycles with a given length for removal may not be an efficient approach. Moreover, for a given value of the memory of the convolutional code, a solution allowing the elimination of all the cycles with a given length may not exist, and the algorithm in [36] is unable to identify the subset of cycles that should be removed in order to eliminate as many harmful objects as possible. In this work we overcome such a limitation and improve over [36] in that:

- we provide new theoretical results on spreading matrices and trapping sets;
- we propose an algorithm which starts from any set of objects as input and tries to minimize their multiplicity. These objects are not necessarily cycles and do not necessarily have the same size;
- the proposed algorithm is not necessarily exhaustive, in that it does not necessarily target all the objects in the set of interest;

- we estimate the complexity of the new algorithm and verify that, based on the aforementioned theoretical results, when targeting cycles, it is significantly faster than the exhaustive search and than that in [36].

We demonstrate the effectiveness of the proposed procedure for several exemplary code constructions with variable code memories via enumeration of the target harmful objects and computer simulations. Such a procedure is shown to be particularly effective for harmful objects that correspond to relatively large combinations of short cycles in the code Tanner graph and on codes with a large multiplicity of harmful objects. Even though the analysis carried out in this paper is limited to regular codes, in principle the proposed algorithm also works for irregular codes. However, irregularity yields an increase of the degrees of freedom in the exponent matrix design which, in turn, leads to a larger number of elementary structures forming the harmful objects. Therefore, the computational complexity of their enumeration would inevitably increase.

We also prove that the average number of cycles per node in the unterminated version of the parity-check matrix can be assessed with the sole knowledge of the spreading matrix and of the list of the corresponding cycles in the underlying block QC-LDPC code, which, in turn, can be determined by means of the code exponent matrix. A similar analysis is carried out in the concurrent work [35]. This yields a significant reduction in the computational effort, as the most costly procedure is carried out on a relatively small matrix, i.e., the exponent matrix of the block QC-LDPC code, compared to the semi-infinite exponent matrix of the QC-SC codes. This is the main reason of the speed-up achieved by the algorithm we propose with respect to that in [36], when it targets sets of cycles. Instead, when trapping sets are targeted by our algorithm, with the aim of reducing their multiplicity as much as possible, the counting algorithm we employ is the one proposed in [14], which is already very efficient. Note that, alternatively, elementary trapping sets can be characterized through edge-coloring techniques exploiting the base matrix, as reported in [37].

We remark that the result of the spreading matrix optimization is tied to the choice of the initial QC-LDPC block code. The focus of this paper is not on the optimization of the original QC-LDPC block code, for which a great amount of literature already exists, but rather on the optimization of the corresponding convolutional version. Thus, in order to show the effectiveness of the proposed algorithm, we stick to block code designs leading to spreading matrices with a simple structure, such as array codes [38] and Tanner codes [22], whose properties are well known and allow us to perform a fair comparison with previous literature.

1.3 Outline of the paper

The paper is organized as follows. In Sect. 2, we introduce the notation used throughout the paper and some basic concepts regarding QC-LDPC block codes and SC-LDPCCs derived from them. In Sect. 3, we focus on edge-spreading matrices and the corresponding cycle properties and describe the proposed algorithm. In Sect. 4, we discuss some code examples and assess the corresponding performance. In Sect. 5 we provide a complexity estimate for our algorithm. Finally, in Sect. 6 we draw some conclusions.

2 Notation and definitions

In this section we introduce the notation and definitions we use throughout the paper, and we present the edge-spreading procedure we consider to obtain QC-SC codes from QC-LDPC block codes.

2.1 QC-LDPC codes

We consider QC-LDPC block codes defined through a parity-check matrix $\mathbf{H}$ in the form of an $m \times n$ array of $N \times N$ circulant permutation matrices (CPMs) or all-zero matrices. We do not consider circulant matrices with weight larger than 1, since large-weight circulants yield short unavoidable cycles [39, 40], which are likely to form harmful objects. We denote a CPM of size $N \times N$ as $\mathbf{I}(p_{i,j})$, $i \in \{0, 1, \ldots, m-1\}$, $j \in \{0, 1, \ldots, n-1\}$, where $p_{i,j} \in \{-\infty, 0, 1, \ldots, N-1\}$ and N is the *lifting degree* of the code. When $p_{i,j} \neq -\infty$, $\mathbf{I}(p_{i,j})$ is obtained from the identity matrix through a cyclic shift of its rows to the left by $p_{i,j}$ positions. Conventionally, we denote the all-zero matrix by $\mathbf{I}(-\infty)$. The code length is $L = nN$. The *exponent matrix* of a code is the $m \times n$ matrix $\mathbf{P}$ containing the values $p_{i,j}$ corresponding to the CPM forming the code parity-check matrix. A bipartite graph $\mathcal{G}(\mathbf{H})$, known as Tanner graph, is associated to any parity-check matrix $\mathbf{H}$ as follows:

- any column of $\mathbf{H}$ corresponds to a variable node;
- any row of $\mathbf{H}$ corresponds to a check node;
- there is an edge between the ith check node and the jth variable node if and only if the (i, j)th entry of $\mathbf{H}$ is 1.

The set of $L = nN$ variable nodes is denoted as $\mathcal{V}$, and the set of mN check nodes is denoted as $\mathcal{P}$. The set of edges is denoted as E. Thus, $\mathbf{H}$ coincides with the bi-adjacency matrix of $\mathcal{G}(\mathbf{H})$ and we also denote $\mathcal{G}(\mathbf{H})$ as $\mathcal{G}(\mathcal{V} \cup \mathcal{P}, E)$.

Definition 1 Two bipartite graphs $\mathcal{G}_1(\mathcal{V}_1 \cup \mathcal{P}_1, E_1)$ and $\mathcal{G}_2(\mathcal{V}_2 \cup \mathcal{P}_2, E_2)$ are isomorphic if there is a bijection $f : \mathcal{V}_1 \cup \mathcal{P}_1 \to \mathcal{V}_2 \cup \mathcal{P}_2$ such that $e_1 = \{v_1, p_1\}$ is an element of E_1 if and only if $e_2 = \{f(v_1), f(p_1)\}$ is an element of E_2.

Definition 2 Equivalently, if $\mathbf{H}_1$ and $\mathbf{H}_2$ are the bi-adjacency matrices of the isomorphic graphs $\mathcal{G}_1(\mathbf{H}_1)$ and $\mathcal{G}_2(\mathbf{H}_2)$, respectively, we say that $\mathbf{H}_1$ and $\mathbf{H}_2$ are graph-isomorphic.

If two parity-check matrices are graph-isomorphic, then also their exponent matrices are graph-isomorphic.

Definition 3 For the sake of conciseness, with a slight mathematical abuse, we say that graph-isomorphic parity-check matrices define *isomorphic codes*.

Isomorphic graphs have the same girth and performance under iterative decoding.

Let us consider the subgraph induced by a subset $\mathcal{D}$ of $\mathcal{V}$ and the corresponding set of neighboring check nodes. We define $\mathcal{E}(\mathcal{D})$ and $\mathcal{O}(\mathcal{D})$ as the set of neighboring check

nodes with even and odd degree in such a subgraph, respectively. The *girth* of $\mathcal{G}(\mathbf{H})$, noted by g, is the length of the shortest cycle in the graph.

Definition 4 An (a, b) absorbing set (AS) is a subset $\mathcal{D}$ of $\mathcal{V}$ of size $a > 0$, with $\mathcal{O}(\mathcal{D})$ of size $b \geq 0$ and with the property that each variable node in $\mathcal{D}$ has strictly fewer neighbors in $\mathcal{O}(\mathcal{D})$ than in $\mathcal{P}\backslash\mathcal{O}(\mathcal{D})$. We say that an (a, b) AS $\mathcal{D}$ is an (a, b) *fully AS (FAS)* if, in addition, all nodes in $\mathcal{V} \setminus \mathcal{D}$ have strictly more neighbors in $\mathcal{P}\backslash\mathcal{O}(\mathcal{D})$ than in $\mathcal{O}(\mathcal{D})$.

For a QC-LDPC block code defined by the parity-check matrix $\mathbf{H}$, a necessary and sufficient condition on the entries of $\mathbf{P}$ for the existence of N cycles of length $2k$ in $\mathcal{G}(\mathbf{H})$ is [41]

$$\sum_{i=0}^{k-1} \left(p_{m_i,n_i} - p_{m_i,n_{i+1}}\right) = 0 \mod N, \tag{1}$$

where $n_k = n_0$, $m_i \neq m_{i+1}$, $n_i \neq n_{i+1}$. In the rest of the paper, we refer to cycles in $\mathcal{G}(\mathbf{H})$ and cycles in $\mathbf{H}$ interchangeably. Hence, for some given values of m and n, and for a fixed value of N, one has to find a matrix $\mathbf{P}$ whose entries do not satisfy (1) for any value of $k < g/2$ and for any possible choice of the row and column indexes m_i and n_i in order to achieve a certain girth g.

2.2 QC-SC codes based on QC-LDPC codes

The edge-spreading procedure we consider to obtain QC-SC codes from QC-LDPC block codes exploits an $m \times n$ *spreading matrix* $\mathbf{B}$ with entries in $\mathbb{Z}_{m_s+1}$, where m_s represents the *memory* of the resulting QC-SC code. The spreading matrix $\mathbf{B}$ can also be represented as an integer *spreading vector* $\mathbf{b}$ of length n, where the ith element (b_i) is obtained by replacing the ith column of $\mathbf{B}$, which is a vector in $\mathbb{Z}_{m_s+1}^m$, with its decimal representation, i.e., $b_i = \sum_{j=0}^{m-1} b_{j,i}(m_s+1)^{m-1-j}$. Conversely, $\mathbf{B}$ can be obtained from $\mathbf{b}$ by replacing each decimal entry with the associated (m_s+1)-ary column vector of length m with entries in $\mathbb{Z}_{m_s+1}$. Without loss of generality, we assume that in any column of $\mathbf{B}$ the most significant symbols are those with the smallest row indices. A straightforward conversion from $\mathbf{B}$ to $\mathbf{b}$ will be shown in Example 1.

A *convolutional exponent matrix* derived from $\mathbf{B}$ has the following form

$$\mathbf{P}_{[0,\infty]} = \begin{bmatrix} \mathbf{P}_0 & & \\ \mathbf{P}_1 & \mathbf{P}_0 & \\ \vdots & \mathbf{P}_1 & \ddots \\ \mathbf{P}_{m_s} & \vdots & \ddots \\ & \mathbf{P}_{m_s} & \ddots \end{bmatrix}, \tag{2}$$

where each matrix $\mathbf{P}_k$, $k \in \{0, 1, \ldots, m_s\}$, has the entry at position (i, j) equal to

$$P_k^{(i,j)} = \begin{cases} p_{i,j} & \text{if} \quad \mathrm{k} = \mathrm{B_{i,j}}, \\ -\infty & \text{if} \quad \mathrm{k} \neq \mathrm{B_{i,j}}, \end{cases} \tag{3}$$

and $B_{i,j}$ is the (i, j)th entry of $\mathbf{B}$. Let us remark that $-\infty$ represents void entries in the convolutional exponent matrix and corresponds to the $N \times N$ all-zero matrix in the

associated binary parity-check matrix. Notice that the entries of $\mathbf{P}_{[0,\infty]}$ which are outside the main diagonals are $-\infty$ and have been omitted from (2) for the sake of readability. The parity-check matrix of the QC-SC code is then obtained as

$$\mathbf{H}_{[0,\infty]} = \begin{bmatrix} \mathbf{H}_0 & & \\ \mathbf{H}_1 & \mathbf{H}_0 & \\ \vdots & \mathbf{H}_1 & \ddots \\ \mathbf{H}_{m_s} & \vdots & \ddots \\ & \mathbf{H}_{m_s} & \ddots \end{bmatrix}, \tag{4}$$

where the appropriate $N \times N$ CPM is substituted for the entries of $\mathbf{P}_{[0,\infty]}$ which have values in the set $\{0, 1, \ldots, N-1\}$, and the $N \times N$ all-zero matrix is substituted for the entries of $\mathbf{P}_{[0,\infty]}$ equal to $-\infty$. The constraint length of the code is defined as $\nu_s = L(m_s + 1)$. Any set of columns of $\mathbf{H}_{[0,\infty]}$ with indexes in $\{jnN, \ldots, (j+1)nN - 1\}$, $j = 0, 1, \ldots$, is named the jth *column block*. $\mathbf{H}_{[\alpha,\beta]}$ represents a terminated version of $\mathbf{H}_{[0,\infty]}$, obtained by considering the columns of the semi-infinite parity-check matrix with indexes in $\{\alpha nN, \ldots, \beta nN - 1\}$, i.e., by considering the consecutive column blocks from the αth to the $(\beta - 1)$th.

Let us introduce a straightforward result on the maximum possible span of a cycle.

Lemma 1 *A cycle with length λ is contained within $\mathbf{H}_{[k,k+\lfloor \frac{\lambda}{4} \rfloor m_s]}$, where k is arbitrary.*

Proof Consider that a cycle with length λ has $\frac{\lambda}{2}$ horizontal edges and $\frac{\lambda}{2}$ vertical edges and any horizontal edge can span at most $m_s + 1$ column blocks of $\mathbf{H}_{[0,\infty]}$. Any cycle spanning the largest number of column blocks of (4) has horizontal edges spanning $m_s + 1$ columns. In this scenario, starting from any of the leftmost nodes, $\frac{\lambda}{2}$ edges are required to reach any of the rightmost nodes, and as many edges are needed to return to the not visited leftmost node. Thus, the number of spanned column blocks is at most

$$N_p = \left\lfloor \frac{\lambda/2}{2} \right\rfloor (m_s + 1) - \left(\left\lfloor \frac{\lambda/2}{2} \right\rfloor - 1 \right) = \left\lfloor \frac{\lambda}{4} \right\rfloor m_s + 1.$$

□

We remark that, for finite n, the girth of regular time-invariant codes, such as those considered in this paper, is upper bounded by a value that does not depend on the code memory [42]. In sight of this, and of the finite circulant size of the starting QC-LDPC code, asymptotic analysis density evolution techniques such as those in [5, 6, 43], which rely on the assumption of infinite girth, may not be the most suited tools to predict the performance of designed codes.

Example 1 Consider the (3, 5)-regular array LDPC block code with the exponent matrix

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 2 & 4 & 1 & 3 \end{bmatrix} \tag{5}$$

and $N = 5$. Consider also the $m_s = 2$ spreading matrix $\mathbf{B}$ with columns with entries in $\mathbb{Z}_3$ corresponding to the spreading vector $\mathbf{b}$, where

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 & 3 & 6 & 21 & 10 \end{bmatrix}. \tag{6}$$

Then the constituent blocks of $\mathbf{P}$ are

$$\mathbf{P}_0 = \begin{bmatrix} 0 & 0 & 0 & - & - \\ 0 & - & - & - & 4 \\ - & 2 & 4 & 1 & - \end{bmatrix}, \mathbf{P}_1 = \begin{bmatrix} - & - & - & - & 0 \\ - & 1 & - & 3 & - \\ 0 & - & - & - & 3 \end{bmatrix},$$

and

$$\mathbf{P}_2 = \begin{bmatrix} - & - & - & 0 & - \\ - & - & 2 & - & - \\ - & - & - & - & - \end{bmatrix},$$

where, for simplicity, $-\infty$ has been expressed as $-$. □

2.3 Exhaustive analysis of spreading matrices

According to the definition given in Sect. 2.2, there are $(m_s + 1)^{mn}$ possible spreading matrices. However, some of them define isomorphic codes. When performing an exhaustive analysis, the size of the search space can be reduced, without loss of completeness, using the following property (from [44]):

Lemma 2 *Let* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *be exponent matrices. If* $\mathbf{P}_1$ *can be obtained by permuting the rows or the columns of* $\mathbf{P}_2$*, or if* $\mathbf{P}_1$ *can be obtained by adding or subtracting* (*modulo N*) *the same constant to all the elements of a row or a column of* $\mathbf{P}_2$*, then the corresponding codes are isomorphic.*

It follows from Lemma 2 that the set of exponent matrices that contain at least one zero in each column represents, without loss of generality, the entire space of exponent matrices. Similarly, it is straightforward to show that the set of spreading matrices containing at least one zero in each column represents, without loss of generality, the entire space of spreading matrices. Each of the m entries of a column of $\mathbf{B}$ can assume values in $\{0, 1, \ldots, m_s\}$, and thus, there are $(m_s + 1)^m$ possible column choices. However, we can remove the m_s^m columns which do not contain any zero entries. It follows that

$$[(m_s + 1)^m - m_s^m]^n \tag{7}$$

spreading matrices cover the whole search space. It is straightforward to notice from (7) that the number of candidate spreading matrices becomes very large as the values of m, n and m_s increase. For this reason, in Sect. 3.2 we propose a novel procedure which

allows one to efficiently distinguish "good" candidates from "bad" candidates. Such an algorithm, based on a *tree-search*, does not exclude any candidate spreading matrix a priori.

3 Methods/experimental

In this section we introduce some theoretical results and propose a novel search algorithm for counting the average number of cycles in QC-SC codes. Moreover, we introduce the spreading matrix search algorithm we use to minimize as much as possible the number of target harmful objects.

3.1 Spreading matrices selection criteria

As mentioned in Sect. 2, trapping sets (and therefore absorbing and fully absorbing sets) often originate from cycles, or clusters of cycles. In this section we prove conditions on the existence of cycles in $\mathbf{H}_{[0,\infty]}$; this allows us to derive the number of equations that must be checked for each candidate spreading matrix in order to verify if it is a good or bad candidate. The goodness of a candidate is measured by the number of harmful objects of the underlying block code that are eliminated in the corresponding QC-SC code.

Definition 5 Let us consider a block code described by $\mathbf{P}$, such that entries

$$(p_{m_0^*,n_0^*}, p_{m_0^*,n_1^*} \dots, p_{m_{k-1}^*,n_{k-1}^*}, p_{m_{k-1}^*,n_k^*})$$

satisfy (1). Then, we say that the code contains a *block cycle* with length $\lambda = 2k$ in the Tanner graph corresponding to the parity-check matrix and we represent such a *block cycle* through a matrix $\mathbf{P}^{\lambda}$, such that

$$p_{i,j}^{\lambda} = \begin{cases} p_{i,j} & \text{if} \quad (\mathrm{i},\mathrm{j}) \in \{(\mathrm{m}_0^*, \mathrm{n}_0^*), (\mathrm{m}_0^*, \mathrm{n}_1^*), \dots, (\mathrm{m}_{\mathrm{k}-1}^*, \mathrm{n}_{\mathrm{k}-1}^*), (\mathrm{m}_{\mathrm{k}-1}^*, \mathrm{n}_{\mathrm{k}}^*)\} \\ -\infty & \text{elsewhere} \end{cases}.$$

Definition 6 The *block-cycle distribution* (or *spectrum*) of $\mathbf{H}_{[0,\mathcal{L}]}$ is denoted as $\mathbf{D}^{\mathcal{L},\Lambda}$ and is a vector such that its ith entry $D_i^{\mathcal{L},\Lambda}$ represents the multiplicity of block cycles with length $2i+4 \leq \Lambda$ in $\mathcal{G}(\mathbf{H}_{[0,\mathcal{L}]})$.

The propositions below easily follow from [31, Theorem 1] and [42, Theorem 1].

Proposition 1 *The number of block cycles of length λ spanning exactly i sections, $i \in \{2,3,\dots,\lfloor \frac{\lambda}{4} \rfloor m_s + 1\}$ is*

$$K_i^{\lambda} = D^{i,\lambda}_{\frac{\lambda-4}{2}} - \sum_{j=1}^{i-1} (i+1-j) K_j^{\lambda}, \tag{8}$$

where $K_1^{\lambda} = D^{1,\lambda}_{\frac{\lambda-4}{2}}$.

Proposition 2 *The average number of block cycles with length λ per node, considering a terminated version of $\mathbf{H}_{[0,\infty]}$, $\mathbf{H}_{[0,\mathcal{L}]}$, is*

$$E_{\lambda,\mathcal{L}} = \frac{\sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s+1} (\mathcal{L}+1-i)K_i^{\lambda}}{\mathcal{L}n}.$$

Lemma 3 *In the case of unterminated parity-check matrix $\mathbf{H}_{[0,\infty]}$, we have that the average number of block cycles with length λ per node is*

$$E_{\lambda} = \frac{\sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s+1} K_i^{\lambda}}{n}. \tag{9}$$

Proof Equation (9) follows from the following chain of equations

$$\begin{aligned} E_{\lambda} &= \lim_{\mathcal{L}\to\infty} \frac{\sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s+1} (\mathcal{L}+1-i)K_i^{\lambda}}{\mathcal{L}n} \\ &= \lim_{\mathcal{L}\to\infty} \frac{\mathcal{L}\sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s+1} K_i^{\lambda} + \sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s+1} (1-i)K_i^{\lambda}}{\mathcal{L}n} \\ &= \frac{\sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s+1} K_i^{\lambda}}{n}. \end{aligned}$$

□

A similar reasoning can be used to compute the average number of (a, b) ASs, $E_{(a,b)}$, which is indeed

$$E_{(a,b)} = \frac{\sum_{i=1}^{L_{(a,b,m_s)}} \kappa_i^{(a,b)}}{n}, \tag{10}$$

where $\kappa_i^{(a,b)}$ is the number of considered ASs spanning exactly i column blocks of the convolutional parity-check matrix, obtained similarly to (8), and $L_{(a,b,m_s)}$ is introduced next. The superscript of the sum in (9) represents the maximum span of the (a, b) ASs under consideration [27, 28] that we call $L_{(a,b,m_s)}$. The latter does not only depend on a and b and m_s, but also on the topology of the considered ASs. Let the shortest path that connects any two variable nodes of an (a, b) AS (or generic trapping set) include at most Σ variable nodes (including the nodes at the start and end of the path). Then, it is possible to compute $L_{(a,b,m_s)} = (\Sigma - 1)m_s + 1$, as shown in [31, Lemma 1]. Σ can be computed by means of Dijkstra's algorithm [45].

From (9) and (10) we notice that the average number of objects per node does not depend on $\mathcal{L}$, but only on the number of objects spanning exactly a certain number of column blocks. Based on the above premises, we have the following results:

Lemma 4 *One, and only one, block cycle (except for trivial replicas) satisfying (1) exists in $\mathbf{P}_{[0,\infty]}$ only if a block cycle in $\mathbf{P}$ exists, satisfying the same equation.*

Proof Let us consider a block cycle with length $\lambda = 2k$ in $\mathbf{P}_{[0,\infty]}$, described by the following equation of type (1)

$$(P_{z_0}^{(m_0,n_0)} - P_{z_1}^{(m_0,n_1)}) + \cdots + (P_{z_{k-1}}^{(m_{k-1},n_{k-1})} - P_{z_0}^{(m_{k-1},n_0)}) = 0 \mod N, \tag{11}$$

where $z_0, z_1, \ldots, z_{k-1} \in \{0, 1, \ldots, m_s\}$ are not necessarily different integers. By the definition of edge spreading, given by (3), $P_{z_0}^{(m_0,n_0)} = p_{m_0,n_0}$, $P_{z_1}^{(m_0,n_1)} = p_{m_0,n_1}$, $\cdots$, $P_{z_{k-1}}^{(m_{k-1},n_{k-1})} = p_{m_{k-1},n_{k-1}}$ and $P_{z_0}^{(m_{k-1},n_0)} = p_{m_{k-1},n_0}$. Therefore, there also exists a block cycle with length $\lambda = 2k$ in $\mathbf{P}$ described by

$$(p_{m_0,n_0} - p_{m_0,n_1}) + \cdots + (p_{m_{k-1},n_{k-1}} - p_{m_{k-1},n_0}) = 0 \mod N.$$

Therefore each block cycle in $\mathbf{P}_{[0,\infty]}$ corresponds to a block cycle in $\mathbf{P}$. Moreover, if $P_{z_j}^{m_i,n_i} \neq -\infty$ for $j \in \{0, 1, \ldots, m_s\}$, then $P_{z_{j'}}^{m_i,n_i} = -\infty$ for all $j' \neq j$. This means that there cannot be more than one cycle in $\mathbf{P}_{[0,\infty]}$ corresponding to the same cycle in $\mathbf{P}$, except for shifted replicas. In fact, given the time-invariant structure of $\mathbf{P}_{[0,\infty]}$, all the block cycles described by

$$(P_{z_0}^{(m_0+sm,n_0+sn)} - P_{z_1}^{(m_0+sm,n_1+sn)}) + \cdots + (P_{z_{k-1}}^{(m_{k-1}+sm,n_{k-1}+sn)} - P_{z_0}^{(m_{k-1}+sm,n_0+sn)}) = 0 \mod N, \tag{12}$$

where $s > 0$, are simple replicas of the block cycle described by (11), since $P^{(m_i+sm,n_i+sn)} = P^{(m_i,n_i)}$, $\forall i, s$.

□

In other words, Lemma 4 states that any block cycle in the convolutional code can be *unambiguously* associated to a block cycle in the block code. However, the converse is not necessarily true, as proved in the following lemma:

Lemma 5 *Consider a block cycle with length λ, described by $\mathbf{P}^{\lambda}$, existing in the Tanner graph $\mathcal{G}(\mathbf{H})$ corresponding to the parity-check matrix of the block QC-LDPC code described by $\mathbf{P}$. Then, after the edge-spreading procedure based on $\mathbf{B}$ is applied, such a block cycle also exists in $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ if and only if $\mathbf{B}^{\lambda}$ satisfies (1) over $\mathbb{Z}$, where*

$$\begin{cases} B_{i,j}^{\lambda} = -\infty & \text{if} \quad \mathrm{p}_{\mathrm{i,j}}^{\lambda} = -\infty, \\ B_{i,j}^{\lambda} = B_{i,j} & \text{otherwise.} \end{cases}$$

Proof Let us consider the matrix $\mathbf{R}$, derived from $\mathbf{P}$, with entries

$$\begin{cases} R_{i,j} = 0 & \text{if} \quad \mathrm{p}_{\mathrm{i,j}} = -\infty, \\ R_{i,j} = 1 & \text{otherwise.} \end{cases}$$

Suppose that a simple cycle $\mathcal{C}$ with length λ exists in $\mathcal{G}(\mathbf{R})$. The spreading operation defined by $\mathbf{B}$ yields a matrix $\mathbf{R}_{[0,\infty]}$ such that $\mathcal{G}(\mathbf{R}_{[0,\infty]})$ will still contain $\mathcal{C}$ if and only if the entries of $\mathbf{B}$ that are in the same positions as the 1's involved in the cycle satisfy (1) over

$\mathbb{Z}$. It is clear that any block cycle in $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ corresponds to a simple cycle in $\mathcal{G}(\mathbf{R}_{[0,\infty]})$ (although the converse, in general, is not true). Since we assumed that $\mathbf{P}^{\lambda}$ describes a block cycle with length λ, $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ will also contain this block cycle if and only if the λ entries of $\mathbf{B}$ that are in the same positions as the λ entries of $\mathbf{P}^{\lambda}$ different from $-\infty$ satisfy (1) over $\mathbb{Z}$. □

Let us suppose that the code defined by an exponent matrix $\mathbf{P}$ contains ν block cycles $\mathbf{P}^{\lambda_0,0},\ldots,\mathbf{P}^{\lambda_{\nu-1},\nu-1}$. Given $\mathbf{B}$, we can extract all the matrices $\mathbf{B}^{\lambda_i,i}$, $i\in\{0,\ldots,\nu-1\}$, that correspond to the block cycles in the QC-LDPC code and check if (1) is satisfied. If so, then the block cycle also exists in the QC-SC code; otherwise, the block cycle does not exist in the QC-SC code. In other words, given an exponent matrix and a spreading matrix, checking as many equations as the number of block cycles in the exponent matrix allows determining the number of block cycles in the convolutional exponent matrix. We also remark that a single block cycle in an exponent matrix corresponds to N cycles in the binary parity-check matrix.

Example 2 Consider the same code and the same spreading matrix as in Example 1 (see (5) and (6), respectively). $\mathcal{G}(\mathbf{H})$ contains twenty block cycles with length $\lambda=6$. For the sake of brevity, we only consider the following three, along with the corresponding entries of the spreading matrix

$$\mathbf{P}^{6,0}=\begin{bmatrix}0&0&-&-&-\\0&-&2&-&-\\-&2&4&-&-\end{bmatrix}\ \mathbf{B}^{6,0}=\begin{bmatrix}0&0&-&-&-\\0&-&2&-&-\\-&0&0&-&-\end{bmatrix},$$
$$\mathbf{P}^{6,1}=\begin{bmatrix}-&0&0&-&-\\0&-&2&-&-\\0&2&-&-&-\end{bmatrix}\ \mathbf{B}^{6,1}=\begin{bmatrix}-&0&0&-&-\\0&-&2&-&-\\1&0&-&-&-\end{bmatrix},$$
$$\mathbf{P}^{6,2}=\begin{bmatrix}-&0&0&-&-\\-&1&-&3&-\\-&-&4&1&-\end{bmatrix}\ \mathbf{B}^{6,2}=\begin{bmatrix}-&0&0&-&-\\-&1&-&1&-\\-&-&0&0&-\end{bmatrix}.$$

Notice that any $\mathbf{P}^{\lambda_i,i}$ complies with (1), as it represents a block cycle in the array LDPC block code. Moreover, (1) is satisfied for $\mathbf{B}^{6,2}$ but not for $\mathbf{B}^{6,0}$ and $\mathbf{B}^{6,1}$. In other words, $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ contains the block cycles of length 6 corresponding to $\mathbf{P}^{6,2}$, but not those associated to $\mathbf{P}^{6,0}$ and $\mathbf{P}^{6,1}$. The same procedure can be applied to test whether the remaining 17 block cycles are also contained in $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ or not. □

The above reasoning permits us to derive an efficient procedure to count the average number of cycles per node in the parity-check matrix of the QC-SC code that is obtained by edge spreading an exponent matrix $\mathbf{P}$ with the edge-spreading matrix $\mathbf{B}$. Such a procedure is described in Algorithm 1 and provides a significant boost with respect to the counting algorithm used in [36].

Algorithm 1

Input exponent matrix $\mathbf{P}$, circulant size N, λ, edge spreading matrix $\mathbf{B}$

1: **procedure** COUNT_AVG_CYCLES($\mathbf{P}$, N, λ, $\mathbf{B}$)
2: $\mathbf{H}_{block} \leftarrow \text{edge\_spread}(\mathbf{P}, \mathbf{B}, N)$
3: $\mathcal{H} \leftarrow \text{list\_cycles}(\mathbf{H}_{block}, \lambda)$
4: $d \leftarrow 0$
5: $i \leftarrow 0$
6: **for** $i \in \{0, \ldots, |\mathcal{H}| - 1\}$ **do**
7: $\mathbf{P}^{\lambda_i,i} \leftarrow \mathcal{H}(i)$
8: **if** $\mathbf{B}^{\lambda_i,i}$ does not satisfy (1) **then**
9: $d \leftarrow d + 1$
10: $i \leftarrow i + 1$
11: goto line 7
12: **end if**
13: **end for**
14: $E_\lambda \leftarrow \frac{|\mathcal{H}|-d}{n}$
15: **end procedure**

Let us suppose that cycles of length λ are the most harmful objects; then, Algorithm 1 returns E_λ. In Algorithm 1, the function edge_spread($\mathbf{P}, \mathbf{B}, N$) performs the edge-spreading procedure as described in Sect. 2.2; the function list_cycles($\mathbf{H}$, λ) lists cycles of length λ of block codes. Clearly, $\mathbf{B}^{\lambda_j,j}$ can be associated to $\mathbf{P}^{\lambda_j,j}$ as described in Lemma 5. Moreover, we have denoted the ith entry of $\mathcal{H}$ as $\mathcal{H}(i)$.

Notice that Lemmas 4 and 5 cannot be straightforwardly extended to general ASs. Let us suppose that an (a, b) AS exists in the parity-check matrix of the initial block code. Let us suppose that none of the cycles contained in the AS are eliminated during the edge-spreading procedure. This is not sufficient to ensure that the (a, b) AS exists in the convolutional version of the matrix, since the edge-spreading procedure may transform it into an (a, b') AS, with $b \neq b'$. So, when the most harmful objects are not cycles, but general (a, b) ASs, we resort to the trapping sets search algorithm proposed in [14] to the convolutional case, considering that the search can be limited to a finite number of column blocks of the parity-check matrix, $L_{(a,b,m_s)}$. Different sets of harmful objects can also be considered. The number of column blocks to be considered is then the largest span for the considered set of harmful objects. Once the considered ASs are enumerated with the algorithm in [14], their average number per node can be obtained using (10).

So, we have devised two different procedures: The first one should be used when the targets of the proposed algorithm are simple cycles; the second one should be used when the target harmful objects are general ASs.

3.2 Design of good QC-SC Codes

In this section we describe a general algorithm, named MInimization of HArmful Objects (MIHAO), which can be applied to an arbitrary harmful object (or set of objects) of interest to find a good QC-SC code. Given the exponent matrix of a QC-LDPC block code, the most harmful objects causing an error rate performance degradation in the associated QC-SC code have to be determined. Such a preliminary analysis depends on the considered decoding algorithm, channel and code, whereas

our algorithm is general and works for any group of objects, regardless of their harmfulness, without requiring the knowledge of the decoding algorithm and the channel. We first describe the proposed optimization algorithm. Then, its application is described in Sect. 4 through some examples including the evaluation of the target harmful objects.

The pseudocode describing the proposed recursive procedure is reported in Algorithm 2. Let us denote the set of target harmful objects as $\mathcal{H} = \{\mathcal{H}_0, \ldots, \mathcal{H}_{\nu-1}\}$. The MIHAO algorithm exploits a tree-based search: The root node of the tree is the all-zero spreading matrix, which characterizes a QC-LDPC block code; the lth tier contains all the spreading matrices with l nonzero entries which reduce the multiplicity of harmful objects with respect to their parent node. If a parent node has no children nodes with better properties than its own, it is discarded, and the algorithm backtracks. Therefore, bad candidates and their children are discarded by the algorithm during the search. If no early stopping criterion is included, after all the survivor candidates are tested, the node corresponding to the spreading matrix yielding the smallest number of harmful objects is the output of the algorithm. On the other hand, some early stopping criteria can be, for example, related to the maximum number of times the algorithm backtracks or the maximum number of tiers it spans.

Algorithm 2

Input exponent matrix $\mathbf{P}$, circulant size N, target harmful objects $\mathcal{H} = \{\mathcal{H}_0, \ldots, \mathcal{H}_{\nu-1}\}$, memory m_s, stopping criterion

Inizialization $\mathbf{B} \leftarrow \mathbf{0}$, $C \leftarrow \frac{|\mathcal{H}|}{n}$

1: **procedure** MIHAO($\mathbf{P}$, N, $\mathcal{H}$, $\mathbf{B}$, m_s, C)
2: $C_{\text{old}} \leftarrow C$
3: $\boldsymbol{\alpha} \leftarrow \text{permute}(\{0, \ldots, m-1\})$
4: $\boldsymbol{\beta} \leftarrow \text{permute}(\{0, \ldots, n-1\})$
5: **while** !Stopping criterion **do**
6: **for** $i \leftarrow 0$ **to** $m-1$ **do**
7: **for** $j \leftarrow 0$ **to** $n-1$ **do**
8: **if** $\mathbf{B}_{\alpha_i, \beta_j} = 0$ **then**
9: **for** $k \leftarrow 0$ **to** m_s **do**
10: $\mathbf{B}_{\text{new}} \leftarrow \mathbf{B}$
11: $\mathbf{B}_{\text{new}}^{(\alpha_i, \beta_j)} \leftarrow k$
12: $C_{\text{new}} \leftarrow \text{count\_avg\_HO}(\mathbf{B}_{\text{new}}, \mathcal{H})$
13: **if** $C_{\text{new}} < C_{\text{old}}$ **then**
14: $\mathbf{B} \leftarrow$ MIHAO($\mathbf{P}$, N, $\mathcal{H}$, $\mathbf{B}_{\text{new}}$, m_s, C_{new})
15: **end if**
16: **end for**
17: **end if**
18: **end for**
19: **end for**
20: **end while**
21: **end procedure**

In Algorithm 2, the function count_avg_HO($\mathbf{B}, \mathcal{H}$) determines the average number of harmful objects per node of the QC-SC code obtained by spreading $\mathbf{P}$ with $\mathbf{B}$. This is accomplished by running lines 4 to 12 of Algorithm 1 if the harmful objects are cycles, and by running the search algorithm in [14] otherwise. Then, the candidate spreading matrices are those maximizing the multiplicity of removed harmful objects.

In fact, the metric we consider to determine whether a candidate is good or bad is the average number of harmful objects per node, as defined above. Note that the algorithm does not guarantee to find the absolute optimal solution, but, as will be shown in Sect. 4 with the help of many examples, it provides better solutions than the best ones available in the literature.

4 Results and discussion

We assess the benefit in terms of structural properties of the codes designed through the proposed approach considering array codes [38] and Tanner codes [22] as a benchmark. Then, we confirm the expected error rate performance improvement via Monte Carlo simulations.

4.1 Optimization results

It is known that the performance of (3, n)-regular array codes is adversely affected by (3, 3) ASs and (4, 2) FASs [12]. It can be shown that (3, 3) ASs and (4, 2) FASs derive from a cycle with length 6 and a cluster of two cycles with length 6, respectively [27]. The results in [28, 29] show that these objects are extremely detrimental also for their QC-SC versions, when used in coded transmissions over the AWGN channel.

Therefore, we have applied Algorithm 2 to minimize their multiplicity in AB QC-SC codes when $m_s = 1$ and $m_s = 2$. Notice that the proposed algorithm, differently from those proposed in [28, 29], has also been used to target harmful objects which are not cycles. The results are shown in Tables 1 and 2, respectively. For

Table 1 Average number of (3, 3) ASs per node $E_{(3,3)}$ and (4, 2) FASs per node $E_{(4,2)}$ in AB SC-LDPC codes with $m = 3, m_s = 1$

$n = N$	5	7	11	13	17	19	23
$E_{(3,3)}$ block code	4	6	10	12	16	18	22
$E_{(3,3)}$ MIHAO	0	0.43	1	1.08	1.88	2.26	3.26
$E_{(3,3)}$ [28, 29]	0	0.43	1	1.23	1.88	2.68	3.78
$E_{(4,2)}$ block code	6	9	15	18	24	27	33
$E_{(4,2)}$ MIHAO	0	0	0	0.15	0.47	0.58	0.52
$E_{(4,2)}$ [28, 29]	0	0	0	0.31	0.53	0.63	0.52

Table 2 Average number of (3, 3) ASs per node $E_{(3,3)}$ and (4, 2) FASs per node $E_{(4,2)}$ in AB SC-LDPC codes with $m = 3, m_s = 2$

$n = N$	7	11	13	17	19	23
$E_{(3,3)}$ block code	6	10	12	16	18	22
$E_{(3,3)}$ MIHAO	0	0	0	0.29	0.42	0.96
$E_{(3,3)}$ [28, 29]	0	0	0	0.24	0.47	0.96
$E_{(4,2)}$ block code	9	15	18	24	27	33
$E_{(4,2)}$ MIHAO	0	0	0	0	0	0.26

$m = 3$, any cycle of length 6 is a (3, 3) AS and, thus, we can feed Algorithm 2 with all the block cycles of length 6 in the AB code. Instead, the search of (4, 2) AS is slightly more complex, as, in this case, Algorithm 2 cannot be fed with simple cycles, but its inputs are combinations of cycles. For (4, 2) FASs of QC-SC codes with $m = 3$, we have that their maximum span is $L_{(4,2,m_s)} = 2m_s + 1$. Notice that the (3, 5)-regular array code is the one considered in Examples 1 and 2. We remark that it is the only (3, n)-regular AB QC-SC code allowing for the removal of all the (3, 3) ASs and (4, 2) FASs when $m_s = 1$.

All the spreading vectors returned by Algorithm 2 are reported in "Appendix 1." Note that, even though the algorithm does not guarantee to find the absolute optimal solution, it provides better solutions than the best ones available in the literature in most cases.

We have also considered the (3, 5)-regular Tanner QC-LDPC code with $L = 155$ and $g = 8$, described by

$$\mathbf{P}_{\frac{2}{5}} = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 5 & 10 & 20 & 9 & 18 \\ 25 & 19 & 7 & 14 & 28 \end{bmatrix}. \tag{13}$$

The dominant trapping sets of this code are known to be (8, 2) ASs [46]. It is shown in [17] that there are 5 different types of (8, 2) ASs in (3, n)-regular codes. All of them contain at least one cycle of length 8. In particular, the (8, 2) ASs identified in [46] as the most dangerous ones for the (3, 5)-regular Tanner code consist of clusters of 15 cycles: 3 of length 8, 4 of length 10, 2 of length 12, 4 of length 14 and 2 of length 16. The simplest approach to eliminate all the (8, 2) ASs consists in removing the shortest cycles they contain, i.e., all the cycles of length 8. In this case, feeding Algorithm 2 with $\mathbf{P}_{\frac{2}{5}}$, $N = 31$, all the cycles of length 8 in $\mathbf{P}_{\frac{2}{5}}$ and $m_s = 1$, we obtain

$$\mathbf{b}_1 = \begin{bmatrix} 2 & 1 & 6 & 1 & 5 \end{bmatrix}.$$

This spreading matrix allows the elimination of all cycles of length 8 and, thus, also $E_{(8,2)} = 0$.

As a further example, we consider the (3, 7)-regular Tanner code with blocklength $L = 301$, $g = 8$ and

$$\mathbf{P}_{\frac{4}{7}} = \begin{bmatrix} 1 & 4 & 16 & 21 & 41 & 35 & 11 \\ 6 & 24 & 10 & 40 & 31 & 38 & 23 \\ 36 & 15 & 17 & 25 & 14 & 13 & 9 \end{bmatrix}. \tag{14}$$

Even though we might have simply assumed that the dominant trapping sets of this code are the same (8, 2) ASs as for the (3, 5)-regular Tanner code, we have analyzed the error patterns returned by the Monte Carlo simulations aimed at estimating the bit error rate (BER) of this block code, over the AWGN channel, using the log-likelihood ratio sum-product algorithm (LLR-SPA) decoder. The results are shown in "Appendix 2" and confirm that the most dangerous objects are indeed (8, 2) ASs, but of a different type than

those considered in [46]. In fact, the (8, 2) ASs which are harmful for the (3, 7)-regular Tanner code are formed by clusters of 14 cycles: 3 of length 8, 4 of length 10, 2 of length 12, 4 of length 14 and 1 of length 16. Again, we note that if we are able to remove all the cycles of length 8 we will also remove all the (8, 2) ASs, whereas the converse is not true, in general. Clearly, removing all the cycles with length 8 is preferable, as they may combine together and form other harmful objects; unfortunately, however, this is not always possible.

For example, the approach of eliminating all the cycles with length 8 for the (3, 7)-regular Tanner code with MIHAO using $m_s = 1$, is not successful. In fact, by performing an exhaustive analysis of the spreading matrices, we have verified that no solution free of cycles with length 8 exists for these parameters. Therefore, since we aim to remove all the (8, 2) ASs, we need to target the cycles composing them, including those with length larger than 8. Notice that this is not equivalent to targeting all the cycles with length 8, but rather a (small) subset of them has to be considered. In order to reach this goal, we need to list the (8, 2) ASs and, for each of them, find the cycles contained in the corresponding subgraph. Consequently, we feed Algorithm 2 with $\mathbf{P}_{\frac{4}{7}}$, $N = 43$, $m_s = 1$, all the (8, 2)-ASs in the parity-check matrix associated to $\mathbf{P}_{\frac{4}{7}}$ and obtain

$$\mathbf{b}_2 = \begin{bmatrix} 3 & 3 & 3 & 1 & 6 & 6 & 5 \end{bmatrix},$$

which indeed eliminates all the (8, 2) ASs.

If, instead, we consider $m_s = 2$, we do not need to target the absorbing sets directly, since, in this case, the MIHAO algorithm can easily find a spreading matrix which removes all the cycles with length 8. In fact, if we feed Algorithm 2 with $\mathbf{P}_{\frac{4}{7}}$, $N = 43$ and all the block cycles of length 8 in $\mathbf{P}_{\frac{4}{7}}$ we obtain

$$\mathbf{b}_3 = \begin{bmatrix} 4 & 11 & 20 & 3 & 13 & 21 & 21 \end{bmatrix},$$

which yields no cycles of length 8 and, thus, no (8, 2) ASs.

We have also considered the code proposed in [31, Example 6], with exponent matrix

$$\mathbf{P}_{\frac{3}{7}} = \begin{bmatrix} 0 & 4 & 5 & 2 & 5 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 6 & 5 \\ 0 & 2 & 4 & 6 & 1 & 3 & 5 \\ 0 & 3 & 6 & 2 & 0 & 0 & 3 \end{bmatrix}. \tag{15}$$

and considered two scenarios:

- $L = 98$, $g = 6$ and spreading vector as in [31], i.e.,

$$\mathbf{b}_4 = \begin{bmatrix} 5 & 9 & 6 & 9 & 6 & 10 & 10 \end{bmatrix}; \tag{16}$$

- $L = 49$, $g = 6$ and spreading vector obtained with MIHAO algorithm, with the aim of minimizing the same objects as in [31] (6 cycles), i.e.,

$$\mathbf{b}_5 = \begin{bmatrix} 55 & 241 & 36 & 73 & 2 & 78 & 84 \end{bmatrix}. \tag{17}$$

The resulting QC-SC codes have $m_s = 1$ and $m_s = 3$, respectively, but the same constraint length $\nu_s = 196$, and are therefore comparable. They will be addressed to as $\mathcal{C}_O$ and $\mathcal{C}_M$, respectively.

4.2 Monte Carlo simulations

In this section we assess the error rate performance of the newly designed Tanner codes in terms of BER via Monte Carlo simulations of binary phase shift keying (BPSK)-modulated transmission over the AWGN channel. All the considered codes are terminated and have block length 120, 000. We have used a sliding window (SW) decoder with window size (in column blocks) $W = 5(m_s + 1)$ performing 100 iterations per window position. The SW decoder performs belief propagation over a window including W blocks of L bits each, after which this window slides forward by L bits before starting over again. For each decoding window position, the SW decoder returns the first L decoded bits, usually called *target bits*, as its output. As in [26–32, 36], we assume that the most harmful objects of a QC-LDPC code are also harmful for its QC-SC version, even when decoded with a large sliding window decoder, such as that used in this paper. This assumption is corroborated by the fact that QC-SC codes have a reduced number of the same harmful object(s) with respect to their block counterparts and also a significantly better performance under sliding window decoding with respect to their block version (see, for example, Fig. 2).

We have first considered the (3, 5)-regular Tanner code and simulated the QC-SC codes obtained by edge-spreading (13) with $\mathbf{b}_1$, reported in Sect. 4.1, and with a random spreading matrix. The results are shown in Fig. 1. The QC-SC code corresponding to $\mathbf{b}_1$ has $g = 10$, whereas that obtained by using a random spreading matrix has $g = 8$ ($E_8 = 0.12$), some residual (8, 2) trapping sets ($E_{(8,2)} = 0.37$) and also a larger multiplicity of cycles with length 10 and 12. While, as expected, the code designed through MIHAO shows improvement over the random code in the region from 0.75 dB to 2 dB, the simulation curves converge at higher signal-to-noise ratios (SNRs) as a result of the remaining dominant objects. So, Fig. 1 is meaningful, since it shows that the QC-SC code suffers the presence of harmful objects which are not immediately deducible from the analysis of the error patterns of the belief propagation decoder for the initial block code, since they are not dominant in that case. In order to improve the performance at a higher SNR, MIHAO should be employed with different target harmful objects, specifically tailored to the SW decoder. An in-depth analysis of this behavior is left for future works, in which the aspects introduced in [47, Section V] should also be accounted for. The error propagation phenomenon suffered by sliding window decoders, as discussed in [48, 49], might also be considered.

We have also simulated the error rate performance of the QC-SC codes obtained by edge spreading the (3, 7)-regular Tanner block code with $\mathbf{b}_2$, which yields a code with $m_s = 1$, and $\mathbf{b}_3$, which yields a code with $m_s = 2$, as detailed in Sect. 4.1. We have compared the performance of these two QC-SC codes with that of the QC-SC codes obtained by edge spreading the (3, 7)-regular Tanner block code with random spreading matrices,

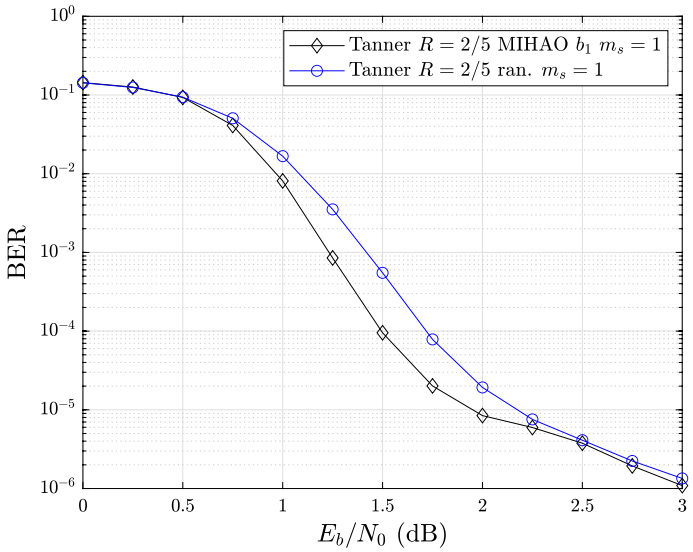

Fig. 1 Simulated performance of (3, 5)-regular Tanner-based QC-SC codes as a function of the signal-to-noise ratio

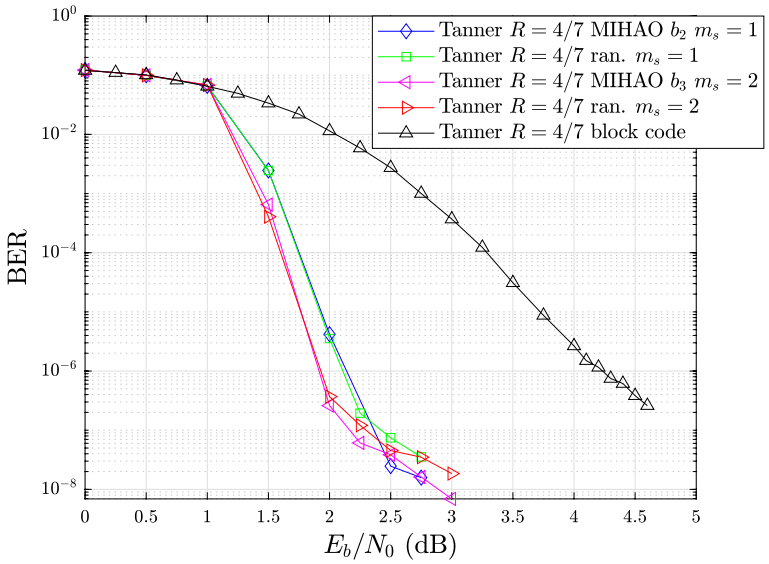

Fig. 2 Simulated performance of (3, 7)-regular Tanner-based QC-SC codes as a function of the signal-to-noise ratio

and the results are shown in Fig. 2. Also in this case, we notice that the increase of the memory yields a gain in the error rate performance and, above all, that the optimization performed by Algorithm 2 improves the error rate performance of the considered codes, especially in the error floor region.

As a final example, we have simulated the error rate performance of the QC-SC codes obtained from (15) in the two scenarios described in Sect. 4.1. The results are

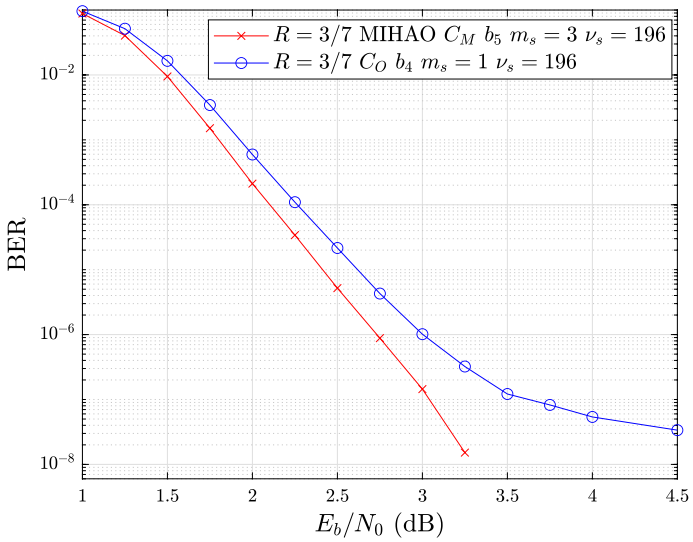

Fig. 3 Simulated performance of (4, 7)-regular QC-SC codes as a function of the signal-to-noise ratio

shown in Fig. 3. Also in this last case, the code obtained by edge spreading $P_{\frac{3}{7}}$ with the spreading matrix obtained through the MIHAO algorithm outperforms that obtained with the spreading matrix proposed in [31], from an error rate performance point of view.

Alternative input QC-LDPC codes of girth 8, with potential performance improvements over Tanner codes, especially in the error floor region, are those proposed in [15, 50–52], among many others. We leave the performance analysis of the corresponding optimized QC-SC codes as a spark for future research.

5 Complexity of the algorithm

In this section, we briefly discuss how the complexity of the proposed procedure grows asymptotically with n and m_s. We also provide complexity estimations in the finite-length regime by considering some examples.

5.1 Asymptotic complexity

Looking at the whole optimization process, we can separate two phases and the corresponding complexity: i) finding the harmful objects in a matrix with finite size (which is either a parity-check matrix or a portion of it) and ii) executing the MIHAO algorithm. Clearly, the first contribution depends on the nature and size of the searched harmful objects. By using the search method proposed in [53], the complexity of the search of trapping sets in a block code with blocklength L, excluding the initial cycle search, increases as $O(L)$. For completeness, we must also consider that the complexity of the search of cycles with length λ, for QC-LDPC codes with column weight equal to

m, increases as $O(m^{\lambda+1})$. Obviously, when this is performed on the block code, m_s does not play any role.

The complexity of the MIHAO algorithm, when targeting cycles, can be estimated as follows. Assuming that picking a value from or changing an entry in a matrix has negligible computational complexity, we can estimate the complexity of the MIHAO algorithm as the complexity of evaluating (1), multiplied by the number of tested spreading matrices (denoted as $N_{\mathbf{B}}$) and multiplied by the number of cycles per test (that is, $|\mathcal{H}|$). The complexity of computing (1) is the same as the cost of the sum of λ elements, where λ is the length of the considered cycle. Conservatively assuming that all the cycles have the same length, i.e., the length of the longest cycle in $\mathcal{H}$, denoted as $\lambda_{\max}$, the complexity of evaluating (1) increases as $O((\lambda_{\max}-1)\log_2(m_s))$. Combining these results, we finally obtain that the complexity of the whole procedure asymptotically increases as

$$O(m^{\lambda_{\max}+1}+\log_2(m_s)|\mathcal{H}|N_{\mathbf{B}}). \tag{18}$$

Note that in (18), $|\mathcal{H}|$ does not depend on m_s, whereas $N_{\mathbf{B}}$ depends on m_s, m, and on the specific code considered.

When, instead, the target harmful objects are general (that is, not cycles) (a, b) ASs, we refer the interested reader to the complexity analysis in [14, Section IV-E], where the blocklength should be substituted with $L_{(a,b,m_s)}L$, being L the blocklength of the starting QC-LDPC code, and $L_{(a,b,m_s)}$ the maximum span (in terms of column blocks) of any harmful object of interest.

5.2 Complexity examples for the finite-length scenario

In order to assess the efficiency of the MIHAO algorithm in a finite-length scenario, we have evaluated the average number of spreading matrices the novel algorithm tests before finding a solution, for the codes with the smallest spreading matrices (so that comparison with exhaustive search is feasible).

Let us consider the (3, 5)-regular array code. Running the MIHAO algorithm with (3, 3) ASs as input, required testing an average of 21 spreading matrices out of the total $8^5 = 32,768$ possible ones (0.06% of the search space) before finding a solution which eliminates such ASs in the QC-SC code with $m_s = 1$, over 1000 trials. A smart exhaustive search on the reduced search space described in Sect. 2.3 found 80 different solutions, out of the 16,807 possible spreading matrices and, therefore, the expected value of tested spreading matrices before finding a valid solution is 211 (1.26% of the search space). If a (non-smart) exhaustive search is performed, the search space has 2^{15} elements, out of which 100 elements have been found to be solutions of the problem of eliminating all (3, 3) ASs. Therefore, the expected percentage of the solution space that has to be spanned to find a solution is 1% (328 spreading matrices). By applying the minimum overlapping (MO) method proposed in [29], the search space contains 1000 spreading matrices. However, this reduced search space does not contain any of the 100 solutions found by means of the exhaustive search.

Let us also consider the (3, 7)-regular array code. Running the MIHAO algorithm 1000 times, with (3, 3) ASs as input, required testing an average of 1, 350 spreading out of

the total 27^7 possible choices before finding a spreading matrix which eliminated all such sets in the QC-SC code with $m_s = 2$ ($1.3 \cdot 10^{-5}$% of the search space). Spanning the entire search spaces of the conventional exhaustive search and the smart exhaustive search is too complex, since they would require testing 3^{21} and 19^7 spreading matrices, respectively, and for this reason the total number of solutions in the respective search spaces is not known.[1] In order to set up a fair comparison with MIHAO, we have randomly generated spreading matrices using the above methods until a solution was found and repeated the experiment 1000 times, counting the average number of tested matrices before finding it. Namely, the exhaustive search requires testing an average of 4227 matrices before finding a solution and the smart exhaustive search 1709 matrices, on average. The MO method considers a reduced search space containing 9261000 spreading matrices. We have verified that it contains 3928 solutions and, therefore, the expected number of spreading matrices to be tested before finding a solution is 2358.

(3, 5) Tanner-based QC-SC codes with $m_s = 1$ and no cycles of length 8 have been found by the MIHAO algorithm by testing an average of 14 spreading matrices out of the total 32768 possible ones (0.04% of the search space), over 1000 attempts. The smart exhaustive search found 540 solutions out of 16807; the expected value of tested matrices before finding a solution is thus 32. The exhaustive search found 570 solutions out of the 2^{15} spreading matrices, therefore needing to test an expected value of 58 matrices before finding a solution. In this case, the MO method considers a reduced space of 1000 spreading matrices containing 45 solutions; therefore, the expected value of spreading matrices to be tested before finding a solution is 23.

Finally, (3, 7) Tanner-based QC-SC codes with $m_s = 2$ and no cycles of length 8 were found by the MIHAO algorithm by testing an average of 5292 spreading matrices out of the total 27^7 possible choices. The smart exhaustive search, the exhaustive search and the MO method would again require testing 19^7, 3^{21} and 9261000 spreading matrices when $m_s = 2$, respectively; these tasks, also due to the increase in the computational complexity of the single test with respect to the previous cases (finding cycles with length 8 is more costly than finding (3, 3) ASs), require a too long processing time and, for this reason, were not performed. Still, just as for the (3, 7)-regular array code, we have randomly searched for solutions in the search spaces of these approaches, stopping when a solution was found. The average number of tested matrices before finding a solution, over 1000 attempts, was 12493, 18396 and 5941, respectively.

Table 3 summarizes the above results. Between round brackets we report the number of spreading matrices in the considered search space and the number of solutions within the considered search space, separated by the symbol /, when it is possible to find them; if the search space is too large to be spanned entirely, we have reported a lower bound on the number of solutions when the algorithms search (3, 3) ASs, and the symbol—if cycles with length 8 are the target. Obviously, for MIHAO we only report the average number of tested matrices before finding a solution, since the novel algorithm stops as soon as it finds a solution. We notice (see bold values) how MIHAO is, in all the

[1] We have empirically found a lower bound for the smart exhaustive search: Its search space contains more than 673 solutions.

Table 3 Average number of tested matrices before finding a solution (between round brackets: number of spreading matrices in the considered search space and number of solutions within this search space), for different codes and search algorithms

Code	m_s	MIHAO	Exhaustive search	Smart exhaustive search	Minimum overlapping [29]
Array (3, 5)	1	**21**	328 (2^{15}/100)	211 (16807/80)	No solution (1000/0)
Tanner (3, 5)	2	**1350**	4227 (3^{21}/–)	1709 ($19^7/\geq$ 673)	2358 (9261000/3928)
Array (3, 7)	1	**14**	58 (2^{15}/570)	32 (16807/540)	23 (1000/45)
Tanner (3, 7)	2	**5292**	18396 (3^{21}/–)	12493 (19^7/–)	5941 (9261000/–)

considered cases, the algorithm requiring the smallest average number of tested matrices before finding a solution.

The speed-up of the novel MIHAO algorithm with respect to its previous version [36], and to the random search (when feasible), can be found in "Appendix 1," referred to array codes. The speed-up is quantified in relative terms and is therefore independent of the specific computer used to obtain the results.

6 Conclusions

We have proposed an efficient algorithm that enables the construction of good QC-SC codes based on QC-LDPC block codes from the perspective of harmful objects. The algorithm is flexible and allows the analysis of codes with different structure and values of memory and rate. The algorithm assumes many classes of harmful objects as the target of a search-and-remove process aimed to eventually optimize codes in terms of error rate performance. Through several examples, we have verified that our algorithm can outperform those available in previous literature, in terms of reduced number of harmful objects, and simulation of the newly designed codes has given evidence of the performance gain achievable. Finally, we have shown with many examples that MIHAO presents advantages in terms of computational complexity over competitive solutions.

Appendix 1

In this appendix, we list all the spreading vectors returned by Algorithm 2 that achieve the results shown in Tables 1 and 2 and the speed-up of the newly proposed algorithm with respect to a random search and to the previous version of the algorithm (the algorithms are implemented in MATLAB and are executed on a standard laptop computer with an Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz and 4 GB RAM). In the speed-up expression, we have denoted the running time of the random search, of the version of the algorithm in [36] and of the proposed version of the algorithm as t_{ran}, $t_{[1]}$ and t_{MIHAO}, respectively. These results are reported in Tables 4 and 5. A time limit of one week has been considered for the running time of the algorithms. When the random search (or the previous version of MIHAO) required more than a week to find the same solution as the newly proposed algorithm, the speed-up cannot be computed and the corresponding entry of the table is a –. It is interesting to note that for most of the considered values of n, the spreading vector yielding the smallest multiplicity of (3, 3) ASs,

Table 4 Output of Algorithm 2 for (3, n) AB SC-LDPC codes, $m_s = 1$and speed-up of the new algorithm

n	5	7	11	13	17
$\mathbf{b} \to E_{(3,3)}$	0, 3, 6, 6, 3	3, 5, 3, 3, 6, 5, 5	2, 1, 1, 3, 4, 4, 2, 4, 4, 1, 1	1, 4, 6, 6, 4, 3, 4, 3, 1, 1, 3, 6, 0	1, 4, 2, 6, 2, 2, 1, 6, 1, 4, 2, 4, 3, 1, 2, 5, 4
$\mathbf{b} \to E_{(4,2)}$	0, 3, 6, 6, 3	3, 5, 3, 3, 6, 5, 5	2, 1, 1, 3, 4, 4, 2, 4, 4, 1, 1	1, 4, 6, 6, 4, 3, 4, 3, 1, 1, 3, 6, 0	1, 4, 2, 6, 2, 2, 1, 6, 1, 4, 2, 4, 3, 1, 2, 5, 4
$\frac{t_{\text{ran}}}{t_{\text{MIHAO}}}$	1.45	1.92	4.11	5.67	–
$\frac{t_{[31]}}{t_{\text{MIHAO}}}$	1.06	0.98	2.06	1.97	6.1

n	19	23
$\mathbf{b} \to E_{(3,3)}$	4, 2, 2, 3, 4, 2, 1, 2, 1, 1, 4, 5, 4, 1 , 1, 2, 1, 2, 4	3, 1, 4, 5, 3, 2, 4, 2, 6, 4, 2, 4, 2, 1, 3, 4, 6, 1, 2, 1, 1, 6, 4
$\mathbf{b} \to E_{(4,2)}$	5, 1, 5, 6, 4, 3, 5, 3, 1, 6, 5, 6, 3, 3 , 2, 2, 5, 1, 7	3, 1, 4, 5, 3, 2, 4, 2, 6, 4, 2, 4, 2, 1, 3, 4, 6, 1, 2, 1, 1, 6, 4
$\frac{t_{\text{ran}}}{t_{\text{MIHAO}}}$	–	–
$\frac{t_{[31]}}{t_{\text{MIHAO}}}$	9.64	–

Table 5 Output of Algorithm 2 for (3, n) AB SC-LDPC codes, $m_s = 2$ and speed-up of the new algorithm

n	7	11	13
$\mathbf{b} \to E_{(3,3)}$	10, 12, 23, 23, 20, 12, 12	11, 7, 15, 5, 15, 7, 11, 19, 11, 11, 19	8, 10, 10, 16, 16, 2, 18, 3, 18, 7, 2, 23, 2
$\mathbf{b} \to E_{(4,2)}$	18, 13, 23, 7, 7, 13, 7	11, 2, 17, 6, 8, 16, 21, 19, 11, 21, 12	15, 6, 20, 20, 1, 3, 1, 1, 3, 1, 20, 20, 6
$\frac{t_{\text{ran}}}{t_{\text{MIHAO}}}$	1.72	–	–
$\frac{t_{[31]}}{t_{\text{MIHAO}}}$	1.08	9.38	13.31

n	17	19
$\mathbf{b} \to E_{(3,3)}$	19, 15, 11, 5, 11, 7, 7, 15, 5, 15, 19, 7, 19, 7, 15, 11, 11	21, 21, 5, 6, 7, 2, 20, 5, 21, 5, 20, 2, 8, 19, 7, 6, 18, 18, 5
$\mathbf{b} \to E_{(4,2)}$	19, 15, 11, 5, 11, 7, 7, 15, 5, 15, 19, 7, 19, 7, 15, 11, 11	21, 21, 5, 6, 7, 2, 20, 5, 21, 5, 20, 2, 8, 19, 7, 6, 18, 18, 5
$\frac{t_{\text{ran}}}{t_{\text{MIHAO}}}$	–	–
$\frac{t_{[31]}}{t_{\text{MIHAO}}}$	–	–

n	23
$\mathbf{b} \to E_{(3,3)}$	11, 5, 5, 7, 11, 11, 7, 15, 5, 15, 7, 19, 7, 19, 19, 7, 19, 11, 15, 19, 11, 11, 15
$\mathbf{b} \to E_{(4,2)}$	11, 5, 5, 7, 11, 11, 7, 15, 5, 15, 7, 19, 7, 19, 19, 7, 19, 11, 15, 19, 11, 11, 15
$\frac{t_{\text{ran}}}{t_{\text{MIHAO}}}$	–
$\frac{t_{[31]}}{t_{\text{MIHAO}}}$	–

denoted as $\mathbf{b} \to E_{(3,3)}$, is also the one yielding the smallest multiplicity of (4, 2) FASs, denoted as $\mathbf{b} \to E_{(4,2)}$.

Appendix 2

In this appendix we show the error patterns that cause the failures of a log-likelihood ratio sum-product algorithm performing 100 iterations on the (3, 7)-regular block Tanner QC-LDPC code considered in Sects. 4.1 and 4.2, with $L = 301$ and $g = 8$, described by the following exponent matrix

$$\mathbf{P}_{\frac{4}{7}} = \begin{bmatrix} 1 & 4 & 16 & 21 & 41 & 35 & 11 \\ 6 & 24 & 10 & 40 & 31 & 38 & 23 \\ 36 & 15 & 17 & 25 & 14 & 13 & 9 \end{bmatrix}.$$

For the sake of brevity, we only give the error patterns for $\frac{E_b}{N_0} = 4.85$ dB, which is the largest $\frac{E_b}{N_0}$ for which we were able to collect a sufficiently large number of error patterns (more than 100) in a reasonable time. In this case, all the decoding failures were caused by the same initial pattern (*inducing* set), whose support (variable node indexes) is

$$\begin{bmatrix} 11 & 22 & 30 & 51 & 112 & 173 & 201 & 205 & 296 \end{bmatrix}.$$

We show its evolution throughout the 100 decoding iterations in Fig. 4. This figure shows that, even though the inducing set is not an (8, 2) AS, the decoder mostly gets stuck at bit positions

$$\begin{bmatrix} 11 & 30 & 51 & 97 & 112 & 173 & 201 & 205 \end{bmatrix}, \tag{19}$$

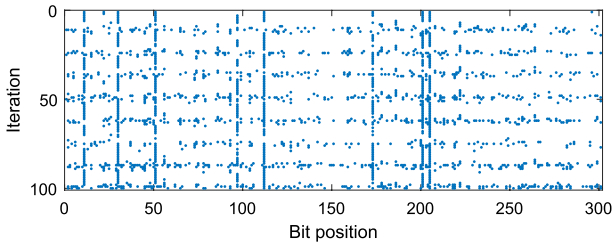

Fig. 4 Evolution of the error pattern causing the failures of the (3, 7)-regular Tanner code for $\frac{E_b}{N_0} = 4.85$ dB. Blue dots represent the positions of errors

which indeed correspond to the most "solid" columns in Fig. 4 and are an (8, 2) AS. Thus, we consider these classes of harmful objects as the most dangerous ones for the (3, 7)-regular Tanner code under the considered decoding algorithm and, therefore, as the input of Algorithm 2. In this case, in order to guarantee results reproducibility, it is important to specify that, for each entry $p_{i,j}$ of $\mathbf{P}_{\frac{4}{7}}$, the corresponding circulant permutation matrix was obtained by cyclic shifts of the rows of the identity matrix toward left by $p_{i,j}$ positions.

Abbreviations

AB	Array-based
AS	Absorbing sets
AWGN	Additive white Gaussian noise
BER	Bit error rate
BPSK	Binary phase shift keying
CPM	Circulant permutation matrix
LDPC	Low-density parity-check
LLR-SPA	Log-likelihood ratio sum-product algorithm
MIHAO	MInimization of harmful objects
QC-LDPC	Quasi-cyclic low-density parity-check
QC-SC codes	Quasi-cyclic spatially coupled LDPC codes from quasi-cyclic low-density parity-check block codes
QC-SC-LDPC	Quasi-cyclic spatially coupled low-density parity-check
SC-LDPCC	Spatially coupled LDPC code
SW	Sliding window

Acknowledgements
The work of Dr. Mitchell is supported by the National Science Foundation under Grant Nos. CCF-2145917 and HRD-1914635. Michele Pacenti acknowledges the support of National Science Foundation under Grants CIF-1855879, CIF-2106189, CCSS-2027844, CCSS-2052751 and CCF-2100013, as well as the support of the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and funded through JPL's Strategic University Research Partnerships (SURP) program.

Author Contributions
All the authors participated in writing the article and revising the manuscript. MB and MP developed the software of the MIHAO algorithm. DM ran the Monte Carlo simulations. MB assessed the complexity comparisons.

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 26 September 2022 Accepted: 6 July 2023
Published online: 25 July 2023

References

1. A. Jiménez Felström, K.S. Zigangirov, Time-varying periodic convolutional codes with low-density parity-check matrix. IEEE Trans. Inf. Theory **45**(6), 2181–2191 (1999)
2. R.G. Gallager, Low-density parity-check codes. IRE Trans. Inform. Theory IT-8, 21–28 (1962)
3. Z. Yang, Y. Fang, G. Zhang, F.C.M. Lau, S. Mumtaz, D.B. da Costa, Analysis and optimization of tail-biting spatially coupled protograph LDPC codes for BICM-ID systems. IEEE Trans. Veh. Technol. **69**(1), 390–404 (2020). https://doi.org/10.1109/TVT.2019.2949600
4. Q. Wang, S. Cai, W. Lin, S. Zhao, L. Chen, X. Ma, Spatially coupled LDPC codes via partial superposition and their application to HARQ. IEEE Trans. Veh. Technol. **70**(4), 3493–3504 (2021). https://doi.org/10.1109/TVT.2021.3065052
5. T.J. Richardson, R.L. Urbanke, The capacity of low-density parity-check codes under message-passing decoding. IEEE Trans. Inf. Theory **47**(2), 599–618 (2001)
6. M. Lentmaier, A. Sridharan, D.J. Costello, K.S. Zigangirov, Iterative decoding threshold analysis for LDPC convolutional codes. IEEE Trans. Inf. Theory **56**(10), 5274–5289 (2010)
7. S. Kudekar, T.J. Richardson, R.L. Urbanke, Threshold saturation via spatial coupling: why convolutional LDPC ensembles perform so well over the BEC. IEEE Trans. Inf. Theory **57**(2), 803–834 (2011)
8. S. Kudekar, T.J. Richardson, R.L. Urbanke, Spatially coupled ensembles universally achieve capacity under belief propagation. IEEE Trans. Inf. Theory **59**(12), 7761–7813 (2013)
9. R. Tanner, A recursive approach to low complexity codes. IEEE Trans. Inf. Theory **27**(5), 533–547 (1981)
10. C. Di, D. Proietti, I.E. Telatar, T.J. Richardson, R.L. Urbanke, Finite-length analysis of low-density parity-check codes on the binary erasure channel. IEEE Trans. Inf. Theory **48**(6), 1570–1579 (2002)
11. T. Richardson, Error floors of LDPC codes, in *Proceedings of 41st Annual Allerton Conference*, Monticello, IL (2003)
12. L. Dolecek, Z. Zhang, V. Anantharam, M.J. Wainwright, B. Nikolic, Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes. IEEE Trans. Inf. Theory **56**(1), 181–201 (2010)
13. Y. Hashemi, A. Banihashemi, On characterization of elementary trapping sets of variable-regular LDPC codes. IEEE Trans. Inf. Theory **60**(9), 5188–5203 (2014)
14. Y. Hashemi, A. Banihashemi, New characterization and efficient exhaustive search algorithm for leafless elementary trapping sets of variable-regular LDPC codes. IEEE Trans. Inf. Theory **62**(12), 6713–6736 (2016)
15. X. Tao, Y. Li, Y. Liu, Z. Hu, On the construction of LDPC codes free of small trapping sets by controlling cycles. IEEE Commun. Lett. **22**(1), 9–12 (2018). https://doi.org/10.1109/LCOMM.2017.2679707
16. R. Asvadi, A.H. Banihashemi, M. Ahmadian-Attari, Lowering the error floor of LDPC codes using cyclic liftings. IEEE Trans. Inf. Theory **57**(4), 2213–2224 (2011)
17. D.V. Nguyen, S.K. Chilappagari, M.W. Marcellin, B. Vasic, On the construction of structured LDPC codes free of small trapping sets. IEEE Trans. Inf. Theory **58**(4), 2280–2302 (2012)
18. Y. Han, W.E. Ryan, Low-floor decoders for LDPC codes. IEEE Trans. Commun. **57**(6), 1663–1673 (2009). https://doi.org/10.1109/TCOMM.2009.06.070325
19. J. Kang, Q. Huang, S. Lin, K. Abdel-Ghaffar, An iterative decoding algorithm with backtracking to lower the error-floors of LDPC codes. IEEE Trans. Commun. **59**(1), 64–73 (2011). https://doi.org/10.1109/TCOMM.2010.101210.090628
20. S. Kang, J. Moon, J. Ha, J. Shin, Breaking the trapping sets in LDPC codes: check node removal and collaborative decoding. IEEE Trans. Commun. **64**(1), 15–26 (2016)
21. D.G.M. Mitchell, M. Lentmaier, D.J. Costello Jr., Spatially coupled LDPC codes constructed from protographs. IEEE Trans. Inf. Theory **61**(9), 4866–4889 (2015)
22. R.M. Tanner, D. Sridhara, A. Sridharan, T.E. Fuja, D.J. Costello, LDPC block and convolutional codes based on circulant matrices. IEEE Trans. Inf. Theory **50**(12), 2966–2984 (2004)
23. A.K. Pradhan, A. Thangaraj, A. Subramanian, Construction of near-capacity protograph LDPC code sequences with block-error thresholds. IEEE Trans. Commun. **64**(1), 27–37 (2016). https://doi.org/10.1109/TCOMM.2015.2500234
24. Y. Fang, S.C. Liew, T. Wang, Design of distributed protograph LDPC codes for multi-relay coded-cooperative networks. IEEE Trans. Wireless Commun. **16**(11), 7235–7251 (2017). https://doi.org/10.1109/TWC.2017.2743699
25. Y. Fang, P. Chen, G. Cai, F.C.M. Lau, S.C. Liew, G. Han, Outage-limit-approaching channel coding for future wireless communications: Root-protograph low-density parity-check codes. IEEE Veh. Technol. Mag. **14**(2), 85–93 (2019). https://doi.org/10.1109/MVT.2019.2903343
26. B. Amiri, A. Reisizadehmobarakeh, H. Esfahanizadeh, J. Kliewer, L. Dolecek, Optimized design of finite-length separable circulant-based spatially-coupled codes: an absorbing set-based analysis. IEEE Trans. Commun. **64**(3), 918–931 (2016)
27. D.G.M. Mitchell, L. Dolecek, D.J. Costello, Absorbing set characterization of array-based spatially coupled LDPC codes, in *Proceedings of IEEE ISIT 2014*, Honolulu, HI, USA, pp. 886–890 (2014)
28. D.G.M. Mitchell, E. Rosnes, Edge spreading design of high rate array-based SC-LDPC codes, in *Proceedings of IEEE ISIT 2017*, Aachen, Germany, pp. 2940–2944 (2017)
29. H. Esfahanizadeh, A. Hareedy, L. Dolecek, A novel combinatorial framework to construct spatially-coupled codes: minimum overlap partitioning, in *Proceedings of IEEE ISIT 2017*, Aachen, Germany, pp. 1693–1697 (2017)
30. A. Beemer, S. Habib, C.A. Kelley, J. Kliewer, A generalized algebraic approach to optimizing SC-LDPC codes, in *Proceedings of 55th Annual Allerton Conference*, Monticello, IL, pp. 672–679 (2017)
31. H. Esfahanizadeh, A. Hareedy, L. Dolecek, Finite-length construction of high performance spatially-coupled codes via optimized partitioning and lifting. IEEE Trans. Commun. **67**(1), 3–16 (2019). https://doi.org/10.1109/TCOMM.2018.2867493
32. A. Hareedy, R. Wu, L. Dolecek, A channel-aware combinatorial approach to design high performance spatially-coupled codes. IEEE Trans. Inf. Theory **66**(8), 4834–4852 (2020). https://doi.org/10.1109/TIT.2020.2979981
33. S. Mo, L. Chen, D.J. Costello, D.G.M. Mitchell, R. Smarandache, J. Qiu, Designing protograph-based quasi-cyclic spatially coupled LDPC codes with large girth. IEEE Trans. Commun. **68**(9), 5326–5337 (2020). https://doi.org/10.1109/TCOMM.2020.3001029

34. L. Schmalen, V. Aref, F. Jardel, Non-uniformly coupled LDPC codes: Better thresholds, smaller rate-loss, and less complexity, in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 376–380 (2017). https://doi.org/10.1109/ISIT.2017.8006553
35. S. Yang, A. Hareedy, R. Calderbank, L. Dolecek, Breaking the computational bottleneck: Probabilistic optimization of high-memory spatially-coupled codes. IEEE Trans. Inf. Theory **69**(2), 886–909 (2023). https://doi.org/10.1109/TIT.2022.3207321
36. M. Battaglioni, F. Chiaraluce, M. Baldi, D.G.M. Mitchell, Efficient search and elimination of harmful objects for the optimization of QC-SC-LDPC codes, in *Proceedings of IEEE GLOBECOM 2019*, Waikoloa, Hawaii, USA (2019)
37. M.-R. Sadeghi, F. Amirzade, Edge-coloring technique to analyze elementary trapping sets of spatially-coupled LDPC convolutional codes. IEEE Commun. Lett. **24**(4), 711–715 (2020). https://doi.org/10.1109/LCOMM.2019.2962671
38. J.L. Fan, Array codes as low-density parity-check codes, in *Proceedings of 2nd Interenational Symposium Turbo Codes*, Brest, France, pp. 543–546 (2000)
39. H. Zhou, N. Goertz, Unavoidable cycles in polynomial-based time-invariant LDPC convolutional codes, in *Proceedings of Wireless Conference on Sustainable Wireless Technology* 2011, Vienna, Austria, pp. 1–6 (2011)
40. Y. Wang, S.C. Draper, J.S. Yedidia, Hierarchical and high-girth QC LDPC codes. IEEE Trans. Inf. Theory **59**(7), 4553–4583 (2013). https://doi.org/10.1109/TIT.2013.2253512
41. M.P.C. Fossorier, Quasi-cyclic low-density parity-check codes from circulant permutation matrices. IEEE Trans. Inf. Theory **50**(8), 1788–1793 (2004)
42. M. Battaglioni, F. Chiaraluce, M. Baldi, M. Lentmaier, Girth analysis and design of periodically time-varying SC-LDPC codes. IEEE Trans. Inf. Theory **67**(4), 2217–2235 (2021). https://doi.org/10.1109/TIT.2021.3059414
43. G. Liva, M. Chiani, Protograph LDPC codes design based on EXIT analysis, in *IEEE GLOBECOM 2007—IEEE Global Telecommunications Conference*, pp. 3250–3254 (2007). https://doi.org/10.1109/GLOCOM.2007.616
44. M. Battaglioni, A. Tasdighi, G. Cancellieri, F. Chiaraluce, M. Baldi, Design and analysis of time-invariant SC-LDPC convolutional codes with small constraint length. IEEE Trans. Commun. **66**(3), 918–931 (2018)
45. E.W. Dijkstra, A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)
46. S. Zhang, C. Schlegel, Causes and dynamics of LDPC error floors on AWGN channels, in *Proceedings of 49th Annual Allerton Conference*, Monticello, IL, pp. 1025–1032 (2011)
47. A. Beemer, C.A. Kelley, Avoiding trapping sets in SC-LDPC codes under windowed decoding, in *Proceedings of ISITA 2016*, Monterey, CA, pp. 206–210 (2016)
48. L. Schmalen, D. Suikat, V. Aref, D. Roesener, On the design of capacity-approaching unit-memory spatially coupled LDPC codes for optical communications, in *ECOC 2016; 42nd European Conference on Optical Communication*, pp. 1–3 (2016)
49. K. Klaiber, S. Cammerer, L. Schmalen, S.t. Brink, Avoiding burst-like error patterns in windowed decoding of spatially coupled LDPC codes, in *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, pp. 1–5 (2018). https://doi.org/10.1109/ISTC.2018.8625312
50. G. Zhang, R. Sun, X. Wang, Several explicit constructions for (3, L) QC-LDPC codes with girth at least eight. IEEE Commun. Lett. **17**(9), 1822–1825 (2013). https://doi.org/10.1109/LCOMM.2013.070913.130966
51. M. Majdzade, M. Gholami, On the class of high-rate QC-LDPC codes with girth 8 from sequences satisfied in GCD condition. IEEE Commun. Lett. **24**(7), 1391–1394 (2020). https://doi.org/10.1109/LCOMM.2020.2983019
52. G. Zhang, Y. Hu, Y. Fang, D. Ren, Relation between GCD constraint and full-length row-multiplier QC-LDPC codes with girth eight. IEEE Commun. Lett. **25**(9), 2820–2823 (2021). https://doi.org/10.1109/LCOMM.2021.3096386
53. Y. Hashemi, A.H. Banihashemi, Characterization and efficient search of non-elementary trapping sets of LDPC codes with applications to stopping sets. IEEE Trans. Inf. Theory **65**(2), 1017–1033 (2019). https://doi.org/10.1109/TIT.2018.2865385

Publisher's Note